A Sandboxing Infrastructure for Alice ML

Bachelor Thesis

Andi Scharfstein, November 2006
Advisor: Andreas Rossberg

Programming Systems Lab
Saarland University

Last Time: Reminder

- Want to have safe environment for execution of dynamically linked code

Last Time: Reminder

- Want to have safe environment for execution of dynamically linked code
- Sandbox provides such an environment

Last Time: Reminder

- Want to have safe environment for execution of dynamically linked code
- Sandbox provides such an environment
- Sandbox mechanism: Wrap language API with code to check if potentially unsafe operations should be allowed

Last Time: Reminder

- Want to have safe environment for execution of dynamically linked code
- Sandbox provides such an environment
- Sandbox mechanism: Wrap language API with code to check if potentially unsafe operations should be allowed
- Allow user to permit specific operations at will

Last Time: Reminder

- Want to have safe environment for execution of dynamically linked code
- Sandbox provides such an environment
- Sandbox mechanism: Wrap language API with code to check if potentially unsafe operations should be allowed
- Allow user to permit specific operations at will

➡ We need:
- definition of „unsafe operations“
- system of „permissions“
- a mechanism to implement API wrapping

Terminology

- *Unsafe operations*: Any operation with access to a system resource

Terminology

- *Unsafe operations*: Any operation with access to a system resource
- *Resources*: File handles, network sockets, CPU time, process IDs...

Terminology

- *Unsafe operations*: Any operation with access to a system resource
- *Resources*: File handles, network sockets, CPU time, process IDs...
- In Alice ML: Resources **always** acquired via *system components*

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java

- Advantages of this approach:

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java
- Advantages of this approach:
  - Checks cannot be circumvented, fundamental part of API

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java
- Advantages of this approach:
  - Checks cannot be circumvented, fundamental part of API
  - High degree of customization

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java
- Advantages of this approach:
  - Checks cannot be circumvented, fundamental part of API
  - High degree of customization
- Disadvantages:

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java
- Advantages of this approach:
  - Checks cannot be circumvented, fundamental part of API
  - High degree of customization
- Disadvantages:
  - Mixes resources and permissions („ugly“)

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java
- Advantages of this approach:
  - Checks cannot be circumvented, fundamental part of API
  - High degree of customization
- Disadvantages:
  - Mixes resources and permissions („ugly“)
  - Checks are performed even when not needed

Inspiration: Java

- Dynamic runtime checks inside API calls first appeared in Java
- Advantages of this approach:
  - Checks cannot be circumvented, fundamental part of API
  - High degree of customization
- Disadvantages:
  - Mixes resources and permissions („ugly“)
  - Checks are performed even when not needed

➡ We strive to achieve a more elegant solution

Java: Example (simplified)

- Every program has an associated *Security Manager*:

Java: Example (simplified)

- Every program has an associated *Security Manager*:
  - Application calls API function
    `file.delete();`

Java: Example (simplified)

- Every program has an associated *Security Manager*:
  - Application calls API function
    ```
    file.delete();
    ```
  - API function checks permissions (and possibly throws SecurityException)
    ```
    SecurityManager sec = System.getSecurityManager();
    sec.checkDelete(file.getName());
    ```

Java: Example (simplified)

- Every program has an associated *Security Manager*:
  - Application calls API function
    ```
    file.delete();
    ```
  - API function checks permissions (and possibly throws SecurityException)
    ```
    SecurityManager sec = System.getSecurityManager();
    sec.checkDelete(file.getName());
    ```
  - If permission is granted, API function deletes file and returns
    ```
    <delete file>
    return true;
    ```

Java: Example (simplified)

- Every program has an associated *Security Manager*:
  - Application calls API function
    ```
    file.delete();
    ```
  - API function checks permissions (and possibly throws SecurityException)
    ```
    SecurityManager sec = System.getSecurityManager();
    sec.checkDelete(file.getName());
    ```
  - If permission is granted, API function deletes file and returns
    ```
    <delete file>
    return true;
    ```
- We use a similar principle, but implement it in a different way than Java

Our Approach: Sample Application

- Consider a browser that can execute programs embedded into websites

Our Approach: Sample Application

- Consider a browser that can execute programs embedded into websites
- These programs are hosted inside a sandbox

Our Approach: Sample Application

- Consider a browser that can execute programs embedded into websites
- These programs are hosted inside a sandbox
- If a program tries to load a system component, the sandbox may deliver a safe counterpart instead. We call this the *substitute component*

Our Approach: Sample Application

- Consider a browser that can execute programs embedded into websites
- These programs are hosted inside a sandbox
- If a program tries to load a system component, the sandbox may deliver a safe counterpart instead. We call this the *substitute component*
- Inside the substitute component, unsafe operations are wrapped with dynamic checks to see if the operation is legal with respect to some ruleset

Component Managers

- Resources can only be acquired by means of system components

Component Managers

- Resources can only be acquired by means of system components
  - ➡ Sufficient to control access to system components

Component Managers

- Resources can only be acquired by means of system components
  - ➡ Sufficient to control access to system components

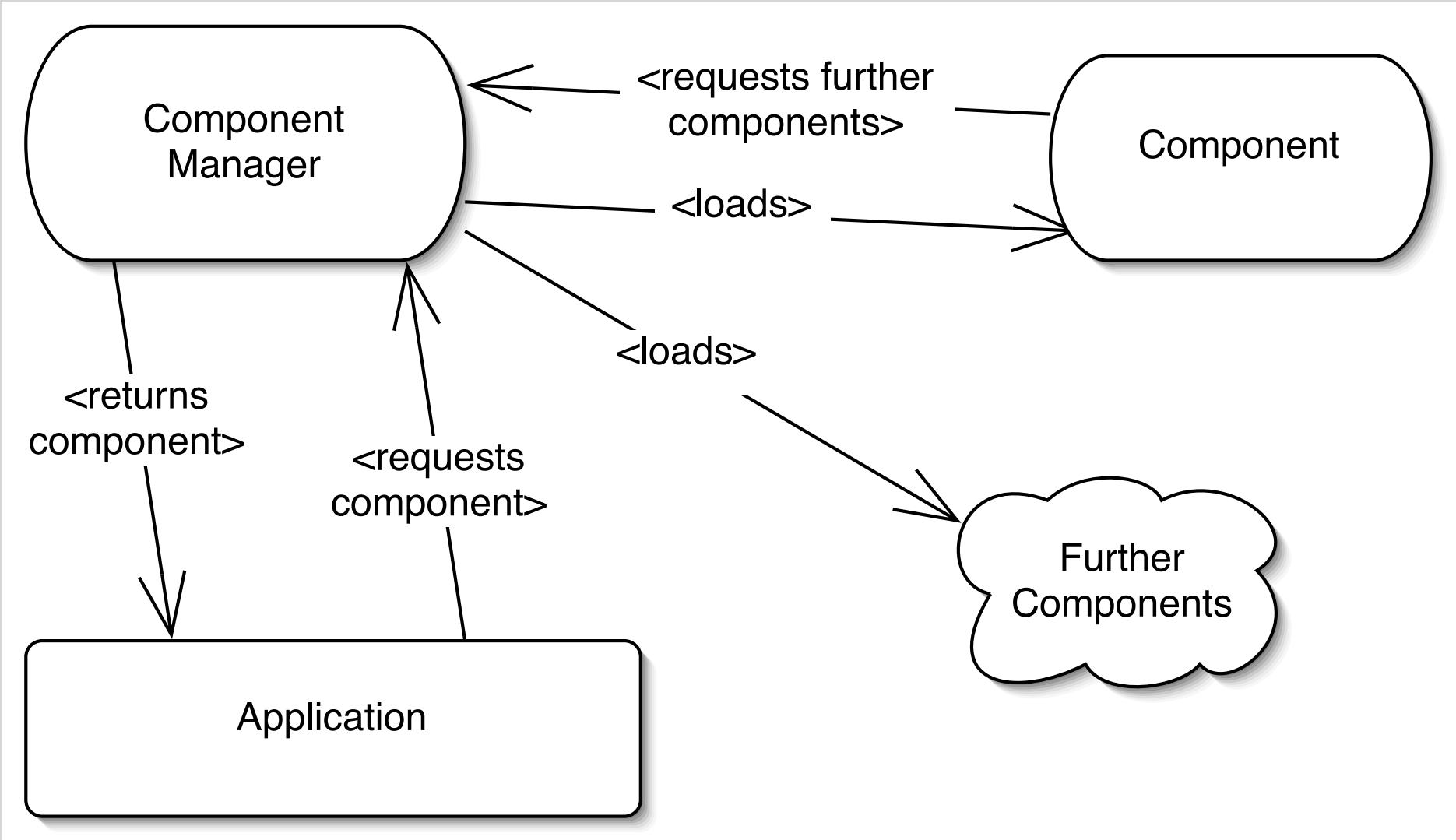
- In Alice ML: Via component managers. These enable easy and elegant sandbox creation

Component Managers

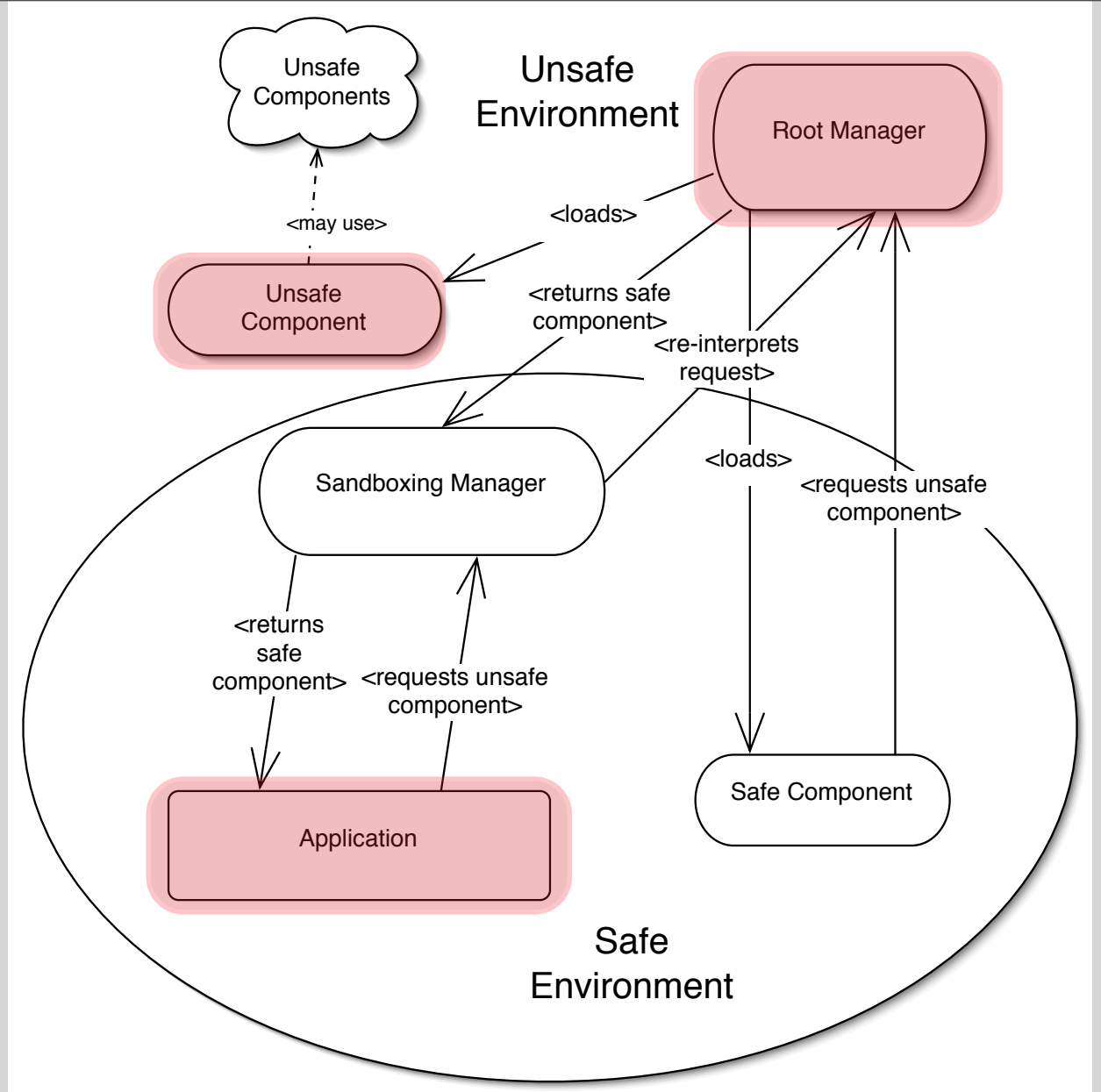
- Resources can only be acquired by means of system components
  - ➡ Sufficient to control access to system components
- In Alice ML: Via component managers. These enable easy and elegant sandbox creation
- A sandbox **is** a (suitably defined) component manager

Component Managers

- Resources can only be acquired by means of system components
  - ➡ Sufficient to control access to system components
- In Alice ML: Via component managers. These enable easy and elegant sandbox creation
- A sandbox **is** a (suitably defined) component manager
- The sandboxing manager makes use of indirection by rewriting component requests



Component Linking



Indirection in sandboxing component mangers

Policies

- Components become safe by behaving according to certain rules („Never write to the file system!“)

Policies

- Components become safe by behaving according to certain rules („Never write to the file system!“)

- We call sets of these rules *policies*

Policies

- Components become safe by behaving according to certain rules („Never write to the file system!“)
- We call sets of these rules *policies*
- Policies can be defined by (dynamically) creating a `Policy` component and adding rules to it

Policies

- Components become safe by behaving according to certain rules („Never write to the file system!“)
- We call sets of these rules *policies*
- Policies can be defined by (dynamically) creating a `Policy` component and adding rules to it
- Policies are a fundamental part of sandboxing manager creation:
  `Sandbox.MkManager(structure Policy : POLICY)`

Policy Rules

- Policy rules are always installed for a single resource (e.g. `fileHandle`)

Policy Rules

- Policy rules are always installed for a single resource (e.g. `fileHandle`)
- They perform two distinct tasks:

Policy Rules

- Policy rules are always installed for a single resource (e.g. `fileHandle`)
- They perform two distinct tasks:
  - Accepting/Rejecting a given value

Policy Rules

- Policy rules are always installed for a single resource (e.g. `fileHandle`)

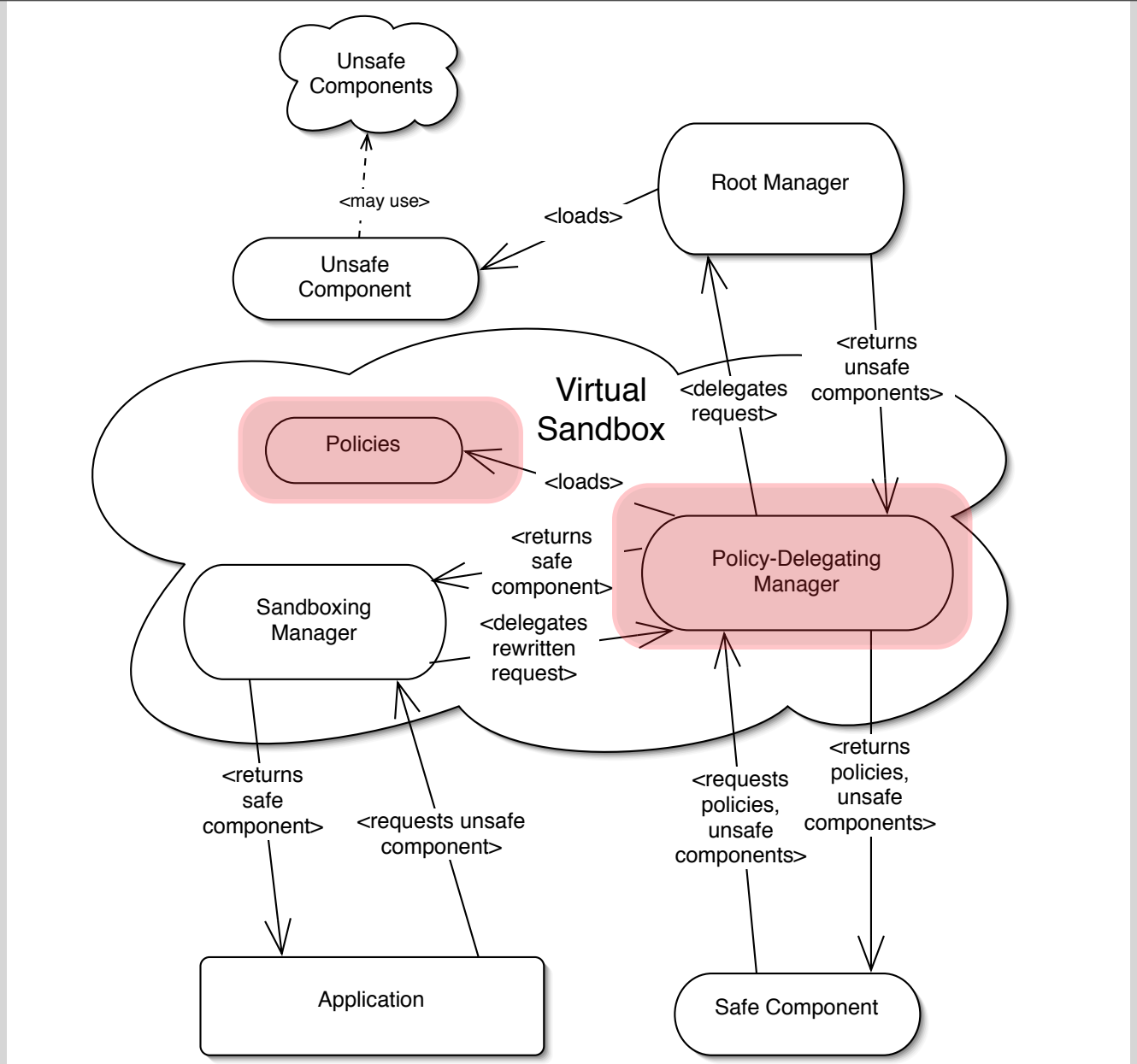
- They perform two distinct tasks:
  - Accepting/Rejecting a given value
  - Rewriting values to guarantee their harmlessness (e.g. prepend a specific directory path to a filename to which the unsafe program has write access: „cookie.txt“ => „~/.safe_storage/cookie.txt“)

Policy Rules

- Policy rules are always installed for a single resource (e.g. `fileHandle`)
- They perform two distinct tasks:
  - Accepting/Rejecting a given value
  - Rewriting values to guarantee their harmlessness (e.g. prepend a specific directory path to a filename to which the unsafe program has write access: „cookie.txt“ => „~/.safe_storage/cookie.txt“)
- Can use nearly arbitrary, user-defined functions for checking and rewriting (return type must be respected)

Policy Rules

- Policy rules are always installed for a single resource (e.g. `fileHandle`)
- They perform two distinct tasks:
  - Accepting/Rejecting a given value
  - Rewriting values to guarantee their harmlessness (e.g. prepend a specific directory path to a filename to which the unsafe program has write access: „cookie.txt“ => „~/.safe_storage/cookie.txt“)
- Can use nearly arbitrary, user-defined functions for checking and rewriting (return type must be respected)
- How can safe substitute components know which rules to follow?



Introducing the Policy-Delegator

Import policies

- Introduce policies for whole system components as well as for single resources

Import policies

- Introduce policies for whole system components as well as for single resources
- These perform a decision for each component import request:

Import policies

- Introduce policies for whole system components as well as for single resources
- These perform a decision for each component import request:
  - Load safe substitute component

Import policies

- Introduce policies for whole system components as well as for single resources
- These perform a decision for each component import request:
  - Load safe substitute component
  - Accept the request without modification

Import policies

- Introduce policies for whole system components as well as for single resources
- These perform a decision for each component import request:
  - Load safe substitute component
  - Accept the request without modification
  - Reject the request (realised by throwing a security error)

Import policies

- Introduce policies for whole system components as well as for single resources
- These perform a decision for each component import request:
  - Load safe substitute component
  - Accept the request without modification
  - Reject the request (realised by throwing a security error)

➡ We don‘t have to insert checks in the API function when they are not needed! Just either accept or load safe substitute component for sandboxing.

Related Work

- Java and .NET offer similar sandboxing approach, but less elegantly

Related Work

- Java and .NET offer similar sandboxing approach, but less elegantly
- Oz inspired Alice components:

Related Work

- Java and .NET offer similar sandboxing approach, but less elegantly
- Oz inspired Alice components:
  - Oz Module Manager ~ Alice Component Manager

Related Work

- Java and .NET offer similar sandboxing approach, but less elegantly
- Oz inspired Alice components:
  - Oz Module Manager ~ Alice Component Manager
  - Oz Functors ~ Alice Components

Related Work

- Java and .NET offer similar sandboxing approach, but less elegantly
- Oz inspired Alice components:
  - Oz Module Manager ~ Alice Component Manager
  - Oz Functors ~ Alice Components
- Similar sandboxing mechanism could in principle be implemented in Oz (component manager delegation features would need to be emulated)

Summary

- Flexible sandboxing mechanism

Summary

- Flexible sandboxing mechanism
- Alice ML enables elegant solution by providing component managers

Summary

- Flexible sandboxing mechanism
- Alice ML enables elegant solution by providing component managers
- Safe substitute components perform the work of regular components while maintaining safety

Summary

- Flexible sandboxing mechanism
- Alice ML enables elegant solution by providing component managers
- Safe substitute components perform the work of regular components while maintaining safety
- Still just „proof of concept“ - Limitations:

Summary

- Flexible sandboxing mechanism
- Alice ML enables elegant solution by providing component managers
- Safe substitute components perform the work of regular components while maintaining safety
- Still just „proof of concept“ - Limitations:
  - Missing pickle verification

Summary

- Flexible sandboxing mechanism
- Alice ML enables elegant solution by providing component managers
- Safe substitute components perform the work of regular components while maintaining safety
- Still just „proof of concept“ - Limitations:
  - Missing pickle verification
  - Language implementation may still contain bugs (e.g. stack overflow)

References (1)

- Andreas Rossberg. The Missing Link - Dynamic Components for ML. 11th International Conference on Functional Programming, Portland, Oregon, USA, ACM Press 2006.

- Drew Dean, Edward Felten and Dan Wallach. Java security: from HotJava to Netscape and beyond. Symposium on Security and Privacy pages 190–200, Oakland, USA, IEEE Computer Society Press, May 1996.

- Li Gong. New Security Architectural Directions for Java. IEEE COMPCON pages 97–102, IEEE Computer Press, 1997.

- Sheng Liang and Gilad Bracha. Dynamic class loading in the Java virtual machine. OOPSLA'98 pages 36–44, 1998.

References (2)

- Robin Milner, Mads Tofte, Robert Harper and David McQueen. Definition of Standard ML (revised). The MIT Press, 1997.

- John H. Reppy and Emden R. Gansner (Editors). The Standard ML Basis Library. Cambridge University Press, October 2004. http://www.standardml.org/Basis/

- Andreas Rossberg, Didier Le Botlan, Guido Tack, Thorsten Brunklaus and Gert Smolka. Alice ML through the looking glass. In Hans-Wolfgang Loidl, editor, Trends in Functional Programming, Vol. 5, Munich, Germany, 2005. Intellect.

- Sun Microsystems. Java 2 Platform SE 5.0 API Documentation. http://java.sun.com/j2se/1.5.0/docs/api/

Appendix: Permission Checking in Java

- Uses global `AccessController` class:

```
FilePermission perm = new FilePermission("path/file",
                                         "read");
AccessController.checkPermission(perm);
```

- This doesn‘t allow for separation of resources (e.g. `"path/file"`) and policies (e.g. `"read"`)

- `checkPermission` uses a thread-local „security context“